The Complexity of Finding Reset Words in Finite Automata

Jörg Olschewski[1,2,*] and Michael Ummels[2,3,†]

[1] Lehrstuhl Informatik 7, RWTH Aachen University, Germany
E-Mail: olschewski@automata.rwth-aachen.de
[2] LSV, CNRS & ENS Cachan, France
[3] Mathematische Grundlagen der Informatik,
RWTH Aachen University, Germany
E-Mail: ummels@logic.rwth-aachen.de

Abstract. *We study several problems related to finding reset words in deterministic finite automata. In particular, we establish that the problem of deciding whether a shortest reset word has length k is complete for the complexity class DP. This result answers a question posed by Volkov. For the search problems of finding a shortest reset word and the length of a shortest reset word, we establish membership in the complexity classes* FP^{NP} *and* $\mathrm{FP}^{NP[\log]}$*, respectively. Moreover, we show that both these problems are hard for* $\mathrm{FP}^{NP[\log]}$*. Finally, we observe that computing a reset word of a given length is FNP-complete.*

1 Introduction

A *synchronising automaton* is a deterministic finite automaton that can be reset to a single state by reading a suitable word. More precisely, we require needs to exist a word w such that, no matter at which state of the automaton we start, w takes the automaton to the same state q; we call any such word w a *reset word* or a *synchronising word*. Although it is easy to decide whether a given automaton is synchronising and to compute a reset word, finding a *shortest* reset word seems to be a hard problem.

The motivation to study reset words does not only come from automata theory: There are applications in the fields of many-valued logics, biocomputing, set theory, and many more [12]. A purely mathematical viewpoint can be obtained by identifying letters with their associated transition functions, which act on a finite set. The task is then to find a composition of these functions such that the resulting function is constant.

* supported by the ESF project GASICS.
† supported by the French project DOTS (ANR-06-SETI-003).

The theory of synchronising automata has been established in the 1960s and is still actively developed. The famous *Černý Conjecture* was formulated in 1971 [3]. The conjecture claims that every synchronising automaton with n states has a reset word of length $(n-1)^2$. As of now, the conjecture has neither been proved nor disproved; the best known upper bound on the length of a reset word is $(n^3-n)/6$, as shown by Pin [8].

While Eppstein [4] showed that the problem of deciding whether there exists a reset word of a given length k is NP-complete, the complexity of deciding whether a *shortest* reset word has length k is not known to be in NP. In his survey paper [12], Volkov asked for the precise complexity of this problem. In this paper, we show that deciding whether a shortest reset word has length k is complete for the class DP, the closure of $\mathrm{NP} \cup \mathrm{coNP}$ under finite intersections. In particular, since every DP-complete problem is both NP-hard and coNP-hard, it is unlikely that the problem of deciding the length of a shortest reset word lies in $\mathrm{NP} \cup \mathrm{coNP}$.[1]

The class DP is contained in the class P^{NP}, i.e. every problem in DP can be solved by a deterministic polynomial-time Turing machine that has access to an oracle for an NP-complete problem. In fact, two oracle queries suffice for this purpose. If one restricts the number of oracle queries to be logarithmic in the size of the input, one arrives at the class $\mathrm{P}^{\mathrm{NP}[\log]}$, which is believed to be a proper superclass of DP. We show that the problem of computing the length of a shortest reset word (as opposed to deciding whether it is equal to a given integer) is, in fact, complete for $\mathrm{FP}^{\mathrm{NP}[\log]}$, the functional analogue of $\mathrm{P}^{\mathrm{NP}[\log]}$. Hence, this problem seems to be even harder than deciding the length of a shortest reset word. Our result complements a recent result by Berlinkov [1], who showed that, unless $\mathrm{P} = \mathrm{NP}$, there is no polynomial-time algorithm that *approximates* the length of a shortest reset word within a constant factor.

For the more general problem of computing a shortest reset word (not only its length), we prove membership in $\mathrm{FP}^{\mathrm{NP}}$, the functional analogue of P^{NP}. While our lower bound of $\mathrm{FP}^{\mathrm{NP}[\log]}$ on computing the length of a shortest reset word carries over to this problem, we leave it as an open problem whether computing a shortest reset word is also $\mathrm{FP}^{\mathrm{NP}}$-hard.

Apart from studying problems related to computing a *shortest* reset word, we also consider the problem of computing a reset word of a given length (represented in unary). We observe that this problem is complete for the class FNP of search problems for which a solution can be verified in polynomial time. In other words: the problem is as hard as computing a satisfying assignment for a given Boolean formula.

[1] We have been informed that Gawrychowski [5] has shown DP-completeness of SHORTEST-RESET-WORD earlier, but his proof has never been published. While his reduction uses a five-letter alphabet, we prove hardness even over a binary alphabet.

2 Preliminaries

Let $\mathcal{A} = \langle Q, \Sigma, \delta \rangle$ be a deterministic finite automaton (DFA) with finite state set Q, finite alphabet Σ and transition function $\delta\colon Q \times \Sigma \to Q$. The transitive closure of δ can be defined inductively by $\delta^*(q, \varepsilon) = q$ and $\delta^*(q, wa) = \delta(\delta^*(q, w), a)$ for each $q \in Q$, $w \in \Sigma^*$ and $a \in \Sigma$. We call any word $w \in \Sigma^*$ such that $|\{\delta^*(q, w) \mid q \in Q\}| = 1$ a *reset word* for $\mathcal{A}$, and we say that $\mathcal{A}$ is *synchronising* if such a word exists. Note that, if w is a reset word for $\mathcal{A}$, then so is xwy for all $x, y \in \Sigma^*$.

We assume that the reader is familiar with basic concepts of complexity theory, in particular with the classes P, NP and coNP. We will introduce the other complexity classes that play a role in this paper on the fly; see Appendix A for formal definitions.

3 Decision Problems

The most fundamental decision problem concerning reset words is to decide whether a given deterministic finite automaton is synchronising. Černý [2] noted that it suffices to check for each pair (q, q') of states whether there exists a word $w \in \Sigma^*$ with $\delta^*(q, w) = \delta^*(q', w)$. The latter property can obviously be decided in polynomial time. The best known algorithm for *computing* a reset word is due to Eppstein [4]: his algorithm runs in time $O(|Q|^3 + |Q|^2 \cdot |\Sigma|)$. Computing a *shortest* reset word, however, cannot be done in polynomial time unless the following decision problems are in P.

SHORT-RESET-WORD: Given a DFA $\mathcal{A}$ and a positive integer k, decide whether there exists a reset word for $\mathcal{A}$ of length k.

SHORTEST-RESET-WORD: Given a DFA $\mathcal{A}$ and a positive integer k, decide whether the minimum length of a reset word for $\mathcal{A}$ equals k.

If the parameter k is given in unary, it is obvious that SHORT-RESET-WORD is in NP. However, even if k is given in binary, this problem is in NP: since every synchronising automaton has a reset word of length $p(|Q|)$ (where p is a low-degree polynomial, e.g. $p(n) = (n^3 - n)/6$), to establish whether there exists a reset word of length k, it suffices to guess a reset word of length $\min\{p(|Q|), k\}$. Eppstein [4] gave a matching lower bound by proving that SHORT-RESET-WORD is also NP-hard.

Regarding SHORTEST-RESET-WORD, Samotij [9] showed that the problem is NP-hard. He also claimed that SHORTEST-RESET-WORD is coNP-hard. However, to prove coNP-hardness, he reduced from the *validity problem* for Boolean formulae in 3CNF (i.e., the problem of deciding whether such a formula is a tautology), which is decidable in polynomial time. (Note that a CNF formula is valid if and

only if each of its clauses contains both a positive and a negative occurrence of a variable). We prove that SHORTEST-RESET-WORD is complete for DP, the class of all languages of the form $L = L_1 \setminus L_2$ with $L_1, L_2 \in \mathrm{NP}$. Since DP is a superclass of both NP and coNP, our result implies hardness for both of these classes. In fact, we show that SHORTEST-RESET-WORD is DP-hard even over a binary alphabet.

Theorem 1. SHORTEST-RESET-WORD is DP-complete.

Proof. It is easy to see that SHORTEST-RESET-WORD belongs to DP: indeed, we can write SHORTEST-RESET-WORD as the difference of SHORT-RESET-WORD and SHORT-RESET-WORD$^-$, where

$$\text{SHORT-RESET-WORD}^- = \{(\mathcal{A}, k+1) \mid (\mathcal{A}, k) \in \text{SHORT-RESET-WORD}\},$$

a problem which is obviously in NP (even if k is given in binary).

It remains to prove that SHORTEST-RESET-WORD is DP-hard. We reduce from the canonical DP-complete problem SAT-UNSAT: given two Boolean formulae φ and ψ (in CNF), decide whether φ is satisfiable and ψ is unsatisfiable. More precisely, we show how to construct (in polynomial time) from a pair (φ, ψ) of Boolean formulae in CNF over propositional variables $X_1, \ldots, X_k$ a synchronising automaton $\mathcal{A}$ over the alphabet $\Sigma = \{0, 1\}$ with the following properties:

1. If φ and ψ are satisfiable, then there exists a reset word of length $k+2$.
2. If φ is satisfiable and ψ is unsatisfiable, then a shortest reset word has length $k+3$.
3. If φ is unsatisfiable, then every reset word has length at least $k+4$.

From 1.–3. we get that φ is satisfiable and ψ is unsatisfiable if and only if a shortest reset word has length $k+3$.

Given formulae $\varphi = C_1 \wedge \ldots \wedge C_n$ and $\psi = D_1 \wedge \ldots \wedge D_n$ where, without loss of generality, φ and ψ have the same number n of clauses, and no propositional variable occurs in both φ and ψ, the automaton $\mathcal{A}$ consists of the states s, t_1, t_2, $p_{i,j}$ and $q_{i,j}$, $i \in \{1, \ldots, n\}$, $j \in \{\bot, \top, 1, \ldots, k\}$; the transitions are depicted in Figure 1: an edge from p to q labelled with $\Sigma' \subseteq \Sigma$ has the meaning that $\delta(p, a) = q$ for each $a \in \Sigma'$. The sets $\Sigma_i^j \subseteq \Sigma$ are defined by $0 \in \Sigma_i^j \Leftrightarrow \neg X_j \in C_i$ and $1 \in \Sigma_i^j \Leftrightarrow X_j \in C_i$, and the sets $\Gamma_i^j \subseteq \Sigma$ are defined by $0 \in \Gamma_i^j \Leftrightarrow \neg X_j \in D_i$ and $1 \in \Gamma_i^j \Leftrightarrow X_j \in D_i$. Hence, e.g. $0 \in \Sigma_i^j$ if we can satisfy the ith clause of φ by setting variable X_j to false.

Clearly, $\mathcal{A}$ can be constructed in polynomial time from φ and ψ. To establish our reduction, it remains to verify 1.–3.

To prove 1., assume that φ and ψ are both satisfiable. Since φ and ψ share no variable, there exists an assignment $\alpha\colon \{X_1, \ldots, X_k\} \to \{\text{true}, \text{false}\}$ that satisfies both φ and ψ. We claim that the word $01w$, where $w = w_1 \ldots w_k \in \{0,1\}^k$ is defined by $w_j = 1 \Leftrightarrow \alpha(X_j) = \text{true}$, resets $\mathcal{A}$ to s. Clearly, $\delta^*(q, w) = s$ for all

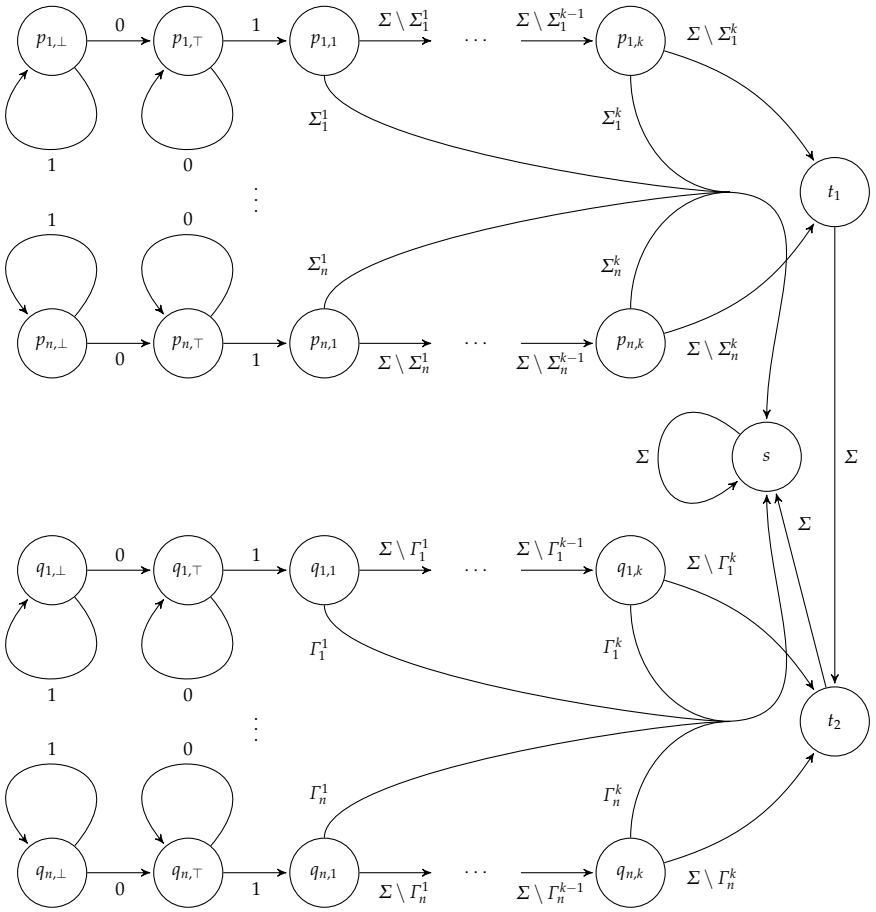

Figure 1. Reducing SAT-UNSAT to SHORTEST-RESET-WORD.

states q that are not of the form $q = p_{i,\bot}$, $q = p_{i,\top}$, $q = q_{i,\bot}$ or $q = q_{i,\top}$. Since $\delta^*(p_{i,\bot}, 01) = \delta^*(p_{i,\top}, 01) = p_{i,1}$ and $\delta^*(q_{i,\bot}, 01) = \delta^*(q_{i,\top}, 01) = q_{i,1}$ for each $i = 1, \ldots, n$, it suffices to show that $\delta^*(p_{i,1}, w) = \delta^*(q_{i,1}, w) = s$ for all i. To prove that $\delta^*(p_{i,1}, w) = s$, consider the least j such that either $X_j \in C_i$ and $\alpha(X_j) = \text{true}$ or $\neg X_j \in C_i$ and $\alpha(X_j) = \text{false}$ (such j exists since α satisfies φ). We have $\delta^*(p_{i,1}, w_1 \ldots w_{j-1}) = p_{i,j}$ and $\delta(p_{i,j}, w_j) = s$ and therefore also $\delta^*(p_{i,1}, w) = s$. The argument for $\delta^*(q_{i,1}, w) = s$ is analogous.

Towards proving 2., assume that φ is satisfiable but ψ is not. Consider an assignment $\alpha \colon \{X_1, \ldots, X_k\} \to \{\text{true}, \text{false}\}$ that satisfies φ. It follows with the same reasoning as above that the word $01w1$, where $w \in \{0,1\}^k$ is defined by $w_j = 1 \Leftrightarrow \alpha(X_j) = \text{true}$, resets $\mathcal{A}$ to s.

To show that a *shortest* reset word has length $k + 3$, it remains to show that there exists no reset word of length $k + 2$. Towards a contradiction, assume that $w = w_1 \ldots w_{k+2}$ is such a word. Note that w resets $\mathcal{A}$ to s and that there

exists $l \geq 2$ such that $\delta^*(q_{i,\perp}, w_1 \dots w_l) = q_{i,1}$ and $\delta^*(q_{i,1}, w_{l+1} \dots w_{k+2}) = s$ for all $i = 1, \dots, n$. Define $\alpha \colon \{X_1, \dots, X_k\} \to \{\text{true}, \text{false}\}$ by setting $\alpha(X_j) = \text{true} \Leftrightarrow w_{l+j} = 1$. Since $l \geq 2$ but $\delta^*(q_{i,1}, w_{l+1} \dots w_{k+2}) = s$, for each i there must exist $j \in \{1, \dots, k\}$ such that $\delta(q_{i,j}, w_{l+j}) = s$. But then either $X_j \in D_i$ and $\alpha(X_j) = \text{true}$ or $\neg X_j \in D_i$ and $\alpha(X_j) = \text{false}$. Hence, α is a satisfying assignment for ψ, contradicting our assumption that ψ is unsatisfiable.

Finally, assume that φ is unsatisfiable. With the same reasoning as in the previous case, it follows that there is no reset word of length $k + 3$. Q.E.D.

The above reduction shows DP-hardness for an alphabet size of $|\Sigma| = 2$. For the special case of only one input letter, note that each reset word is of the form 1^n for some n. Asking whether there exists a reset word of length k thus collapses to the question whether 1^k is a reset word for $\mathcal{A}$. This property can be decided with logarithmic space. Hence, both problems, SHORT-RESET-WORD and SHORTEST-RESET-WORD, are in LOGSPACE for $|\Sigma| = 1$.

4 Search problems

In this section, we leave the realm of decision problems and enter the (rougher) territory of search problems, where the task is not only to decide whether a reset word of some length exists, but to compute a suitable word (or its length). More precisely, we deal with the following search problems:

- Given a DFA $\mathcal{A}$ and a positive integer k in unary, compute a reset word for $\mathcal{A}$ of length k.
- Given a DFA $\mathcal{A}$, compute the length of a shortest reset word for $\mathcal{A}$.
- Given a DFA $\mathcal{A}$, compute a shortest reset word for $\mathcal{A}$.

Let us start with the first problem of computing a reset word of a given length. It turns out that this problem is complete for the class FNP of search problems where the underlying binary relation is both polynomially balanced and decidable in polynomial time.

Proposition 2. The problem of computing a reset word of a given length is FNP-complete.

Proof. Membership in FNP follows from the fact that the binary relation

$$\{((\mathcal{A}, 1^k), w) \mid w \text{ is a reset word for } \mathcal{A} \text{ of length } k\}$$

is polynomially balanced and polynomial-time decidable.

To prove hardness, we reduce from FSAT, the problem of computing a satisfying assignment for a given Boolean formula in conjunctive normal form. To this end, we describe two polynomial-time computable functions f and g, where f computes from a CNF formula φ a synchronising automaton $\mathcal{A} = f(\varphi)$ over

the alphabet $\{0,1\}$ and a unary number $k \in \mathbb{N}$, and g computes from φ and $w \in \Sigma^*$ an assignment for φ, such that, if w is a reset word for $\mathcal{A}$ of length k, then the generated assignment satisfies φ.

Eppstein [4] showed how to compute in polynomial time, given a CNF formula $\varphi = C_1 \wedge \ldots \wedge C_n$ over the variables $X_1, \ldots, X_k$, an automaton $\mathcal{A}_\varphi$ over the alphabet $\{0,1\}$ with the following two properties:

1. A word $w = w_1 \cdots w_k$ is a reset word for $\mathcal{A}$ if and only if the assignment α, defined by $\alpha(X_j) = \text{true} \Leftrightarrow w_j = 1$, satisfies φ.
2. An assignment $\alpha\colon \{X_1, \ldots, X_k\} \to \{\text{true}, \text{false}\}$ satisfies φ if and only if the word $w \in \{0,1\}^k$, defined by $w_j = 1 \Leftrightarrow \alpha(X_j) = \text{true}$, is a reset word for $\mathcal{A}$.

(Note that the reduction we use to prove Theorem 1 has similar properties and could also be used.)

Hence, we can choose f to be the function that maps φ to $(\mathcal{A}_\varphi, 1^k)$ and g to be the function that maps (φ, w) to the corresponding assignment α. (If $|w| \neq k$, then α can be chosen arbitrarily.) Q.E.D.

Remark 3. Note that the mapping $f : \{0,1\}^k \to \{\text{true}, \text{false}\}^{\{X_1,\ldots,X_k\}}$, defined by $f(w)(X_j) = \text{true} \Leftrightarrow w_j = 1$, is a bijection. Eppstein's reduction shows that one can compute from a Boolean formula φ over the variables $\{X_1, \ldots, X_k\}$ an automaton $\mathcal{A}$ such that f remains a bijection when one restricts the domain to reset words for $\mathcal{A}$ and the range to assignments that satisfy φ. Therefore, his reduction can be viewed as a parsimonious reduction from #SAT, the problem of counting all satisfying assignments of a given Boolean formula, to the problem of counting all reset words of a given length (represented in unary). Since the first problem is complete for #P [11], the second problem is #P-hard. On the other hand, it is easy to see that the second problem is in #P. Hence, this problem is #P-complete.

Next, we consider the problem of computing the *length* of a shortest reset word for a given automaton: we establish that this problem is complete for the class $\text{FP}^{\text{NP[log]}}$ of all problems that are solvable by a polynomial-time algorithm with access to an oracle for a problem in NP where the number of queries is restricted to $O(\log n)$.

Theorem 4. The problem of computing the length of a shortest reset word is $\text{FP}^{\text{NP[log]}}$-complete.

Proof. To prove membership in $\text{FP}^{\text{NP[log]}}$, consider Algorithm 1 which is a binary-search algorithm for determining the length of a shortest reset word for an automaton $\mathcal{A}$ with n states. The algorithm is executed in polynomial time: the while loop is repeated $O(\log n)$ times and asks $O(\log n)$ queries to the oracle, which is used for determining whether $\mathcal{A}$ has a reset word of a given length.

Krentel [6] showed that MAX-SAT-SIZE, the problem of computing the maximum number of simultaneously satisfiable clauses of a CNF formula, is complete

Algorithm 1. Computing the length of a shortest reset word.

if $\mathcal{A}$ is not synchronising **then reject**
$low := -1$
$high := (n^3 - n)/6$
while $high - low > 1$ **do**
  $k := \lceil (low + high)/2 \rceil$
  if $\mathcal{A}$ has a reset word of length k **then**
    $high := k$
  else
    $low := k$
end while
return $high$

for $\mathrm{FP}^{\mathrm{NP[log]}}$. Therefore, to establish $\mathrm{FP}^{\mathrm{NP[log]}}$-hardness, it suffices to give a reduction from MAX-SAT-SIZE to our problem. Such a reduction consists of two polynomial-time computable functions f and g with the following properties: f computes from a CNF formula φ a (synchronising) automaton $\mathcal{A} = f(\varphi)$, and g computes from φ and $l \in \mathbb{N}$ a new number $g(\varphi, l) \in \mathbb{N}$ such that, if l is the length of a shortest reset word for $\mathcal{A}$, then the maximum number of simultaneously satisfiable clauses in φ equals $g(\varphi, k)$.

Given a formula $\varphi = C_1 \wedge \ldots \wedge C_n$ over propositional variables $X_1, \ldots, X_k$, the resulting automaton $\mathcal{A}$ is depicted in Figure 2: The input alphabet is $\Sigma := \{0, 1, \$\}$, and the sets $\Sigma_i^j \subseteq \Sigma$ are defined as in the proof of Theorem 1; we set $\lambda := k + n(n+4)$. The behaviour of the transition function on vertices of the form $r_{i,j}$ is defined as follows:

- $\delta(r_{i,j}, \$) = p_{i,1}$ for all $j \in \{-2, \ldots, n+1\}$;
- $\delta(r_{i,j}, 1) = r_{i,j+1}$, $\delta(r_{i,j}, 0) = r_{i,-2}$ for all $j \in \{-2, -1, i\}$;
- $\delta(r_{i,j}, 1) = r_{i,-2}$, $\delta(r_{i,j}, 0) = r_{i,j+1}$ for all $j \in \{0, \ldots, i-1, i+1, \ldots, n\}$;
- $\delta(r_{i,n+1}, 1) = r_{i,-2}$, $\delta(r_{i,n+1}, 0) = s$.

It is not difficult to see that $\mathcal{A}$ can be constructed in polynomial time from φ. Moreover, we claim that, for each $m \in \{0, 1, \ldots, n\}$, there exists an assignment that satisfies at least $n - m$ clauses of φ if and only if $\mathcal{A}$ has a reset word of length $1 + \lambda + k + m(n+4)$. Hence, if l is the length of a shortest reset word for $\mathcal{A}$, then the maximal number of simultaneously satisfiable clauses of φ is given by $n - \left\lceil \frac{\max\{0, l-1-\lambda-k\}}{n+4} \right\rceil$. Clearly, this number can be computed in polynomial time from φ and l.

($\Rightarrow$) Assume that $\alpha \colon \{X_1, \ldots, X_k\} \to \{\text{true}, \text{false}\}$ is an assignment that satisfies all clauses except (possibly) the clauses $C_{i_1}, \ldots, C_{i_m}$, and consider the word

$$w := \$1^\lambda x_1 \ldots x_k z_{i_1} \ldots z_{i_m},$$

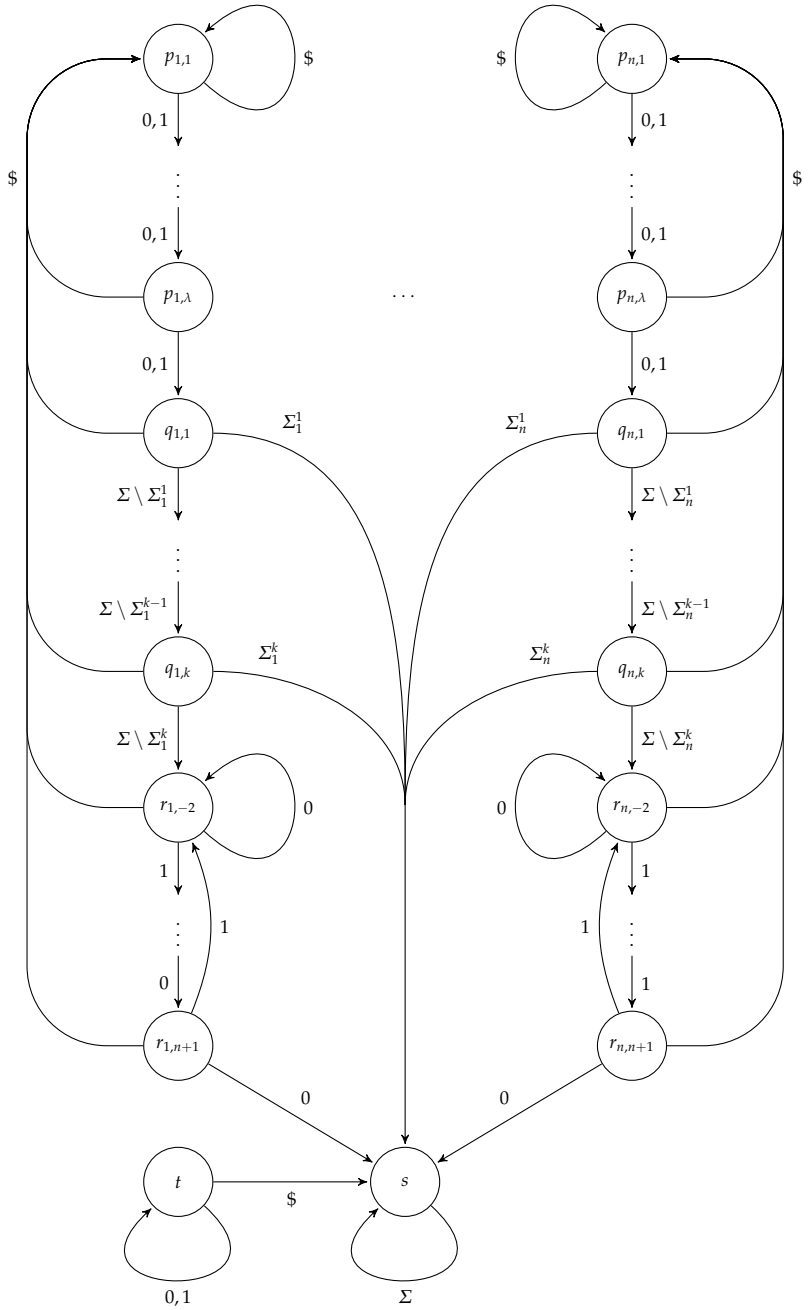

Figure 2. Reducing MAX-SAT-SIZE to computing the length of a shortest reset word.

where $z_i = 110^i10^{n-i+1} \in \{0,1\}^{n+4}$ for $i \in \{1,\ldots,n\}$ and

$$x_j := \begin{cases} 1 & \text{if } \alpha(X_j) = \text{true}, \\ 0 & \text{otherwise.} \end{cases}$$

Note that w has length $1+\lambda+k+m(n+4)$. We claim that w resets $\mathcal{A}$ to s. Since reading \$ has the effect of going from each state of the form $p_{i,j}$, $q_{i,j}$ or $r_{i,j}$ to $p_{i,1}$ and from t to s, and reading 1^λ has the effect of going from $p_{i,1}$ to $q_{i,1}$, it suffices to show that $\delta^*(q_{i,1}, x_1 \ldots x_k z_{i_1} \ldots z_{i_m}) = s$. If C_i is satisfied by α, then this follows from the fact that there exists j such that $\delta(q_{i,j}, x_j) = s$. Otherwise, we have $\delta^*(q_{i,1}, x_1 \ldots x_k) = r_{i,-2}$, $\delta^*(r_{i,-2}, z_j) = r_{i,-2}$ for all $j \neq i$, but $\delta^*(r_{i,-2}, z_i) = s$. Since $i \in \{i_1,\ldots,i_m\}$, this implies that $\delta^*(q_{i,1}, x_1 \ldots x_k z_{i_1} \ldots z_{i_m}) = s$.

($\Leftarrow$) Assume that $\mathcal{A}$ has a reset word of length $1+\lambda+k+m(n+4)$, and let w be a shortest reset word for $\mathcal{A}$. We claim that w has the form $w = \$u$ or $w = u\$$ for $u \in \{0,1\}^*$. Otherwise, $w = u\$v$ for $u, v \in \Sigma^+$. Towards a contradiction, we distinguish the following two cases: $|u| \leq \lambda$ and $|u| > \lambda$. If $|u| \leq \lambda$, then $\delta^*(p_{i,1}, u\$) = p_{i,1}$ for all $i = 1,\ldots,n$, and the word $\$v$ would be a shorter reset word than w. Now assume that $|u| > \lambda$. It must be the case that $\delta^*(p_{i,1}, u) \neq s$ for some $i \in \{1,\ldots,n\}$ because otherwise $\$u$ would be a shorter reset word than w. But then $\delta^*(p_{i,1}, u\$) = p_{i,1}$. Hence, since w resets $\mathcal{A}$ to s and the shortest path from $p_{i,1}$ to s has length greater than λ, $|v| > \lambda$ and $|w| > 1+2\lambda \geq 1+\lambda+k+n(n+4) \geq 1+\lambda+k+m(n+4)$, a contradiction.

Now, if φ is satisfiable, we are done. Otherwise, let us fix $u \in \{0,1\}^*$ such that $w = \$u$ or $w = u\$$. Since φ is not satisfiable, $|u| \geq \lambda + k$. Let $u = yx_1 \ldots x_k z$ where $y, z \in \{0,1\}^*$, $|y| = \lambda$, and $x_j \in \{0,1\}$ for all $j = 1,\ldots,k$. Now consider the assignment α defined by

$$\alpha(X_j) = \begin{cases} \text{true} & \text{if } x_j = 1, \\ \text{false} & \text{otherwise.} \end{cases}$$

Moreover, let

$$I := \{i \in \{1,\ldots,n\} \mid C_i \text{ is not satisfied by } \alpha\}.$$

We claim that $|I| \leq m$ (so α satisfies at least $n-m$ clauses of φ). To see this, first note that $\delta^*(p_{i,1}, yx_1 \ldots x_k) = r_{i,-2}$ for all $i \in I$. Hence, we must have that $\delta^*(r_{i,-2}, z) = s$ for all such i. By the construction of $\mathcal{A}$, this is only possible if z contains the word 110^i10^{n-i+1} as an infix for each $i \in I$. Since these infixes cannot overlap, $|z| \geq |I| \cdot (n+4)$. On the other hand, since $|u| \leq \lambda+k+m(n+4)$, we must have $|z| \leq m(n+4)$. Hence, $|I| \leq m$. Q.E.D.

The construction we have presented to prove Theorem 4 uses a three-letter alphabet. With a little more effort, we can actually reduce the alphabet to an alphabet with two letters 0 and 1: For each state $q \notin \{s,t\}$ of $\mathcal{A}$, there are three states $(q,0)$, $(q,1)$ and $(q,2)$ in the new automaton $\mathcal{A}'$. Additionally, $\mathcal{A}'$ contains the states $(t,0)$, $(t,1)$ and s. The new transition function δ' is defined as follows:

$$\begin{aligned}
&\delta'((q,0),0) = (q,1), && \delta'((q,0),1) = (q,2),\\
&\delta'((q,1),0) = (q,1), && \delta'((q,1),1) = (\delta(q,\$),2),\\
&\delta'((q,2),0) = (\delta(q,0),0), && \delta'((q,2),1) = (\delta(q,1),0)
\end{aligned}$$

for all $q \notin \{s,t\}$, and

$$\begin{aligned}
&\delta'((t,0),0) = s, && \delta'((t,0),1) = (t,1),\\
&\delta'((t,1),0) = (t,0), && \delta'((t,1),1) = (t,1),\\
&\delta'(s,0) = s, && \delta'(s,1) = s.
\end{aligned}$$

Intuitively, taking a transition in $\mathcal{A}$ corresponds to taking two transitions in $\mathcal{A}'$. It is not difficult to see that a shortest reset word for $\mathcal{A}'$ has length $2l$ if a shortest reset word for $\mathcal{A}$ has length l.

For the potentially harder problem of computing a shortest reset word (not only its length), we can only prove membership in $\mathrm{FP}^{\mathrm{NP}}$, the class of all search problems that are solvable in polynomial time using an oracle for a problem in NP (without any restriction on the number of queries). Of course, hardness for $\mathrm{FP}^{\mathrm{NP}[\log]}$ carries over from our previous result. We have not been able to close the gap between the two bounds. To the best of our knowledge, the same situation occurs e.g. for MAX-SAT, where the aim is to find an assignment of a given Boolean formula that satisfies as many clauses as possible.

Theorem 5. The problem of computing a shortest reset word is in $\mathrm{FP}^{\mathrm{NP}}$ and hard for $\mathrm{FP}^{\mathrm{NP}[\log]}$.

Proof. To prove membership in $\mathrm{FP}^{\mathrm{NP}}$, consider Algorithm 2 for computing a shortest reset word for an automaton $\mathcal{A}$ over any finite alphabet Σ. The algorithm obviously computes a reset word of length l, which is the length of a shortest reset word. To see that the algorithm runs in polynomial time if it has access to an NP oracle, note that deciding whether $\mathcal{A}$ has a reset word of a given length with a given prefix is in NP (since a nondeterministic polynomial-time algorithm can guess such a word). Moreover, as we have shown above, computing the *length* of a shortest reset word can be done by a polynomial-time algorithm with access to an NP oracle.

Hardness for $\mathrm{FP}^{\mathrm{NP}[\log]}$ follows from Theorem 4 since the problem of computing the length of a shortest reset word is trivially reducible to the problem of

Algorithm 2. Computing a shortest reset word.

if $\mathcal{A}$ is not synchronising **then reject**
Compute the length l of a shortest reset word for $\mathcal{A}$
$w := \varepsilon$
while $|w| < l$ **do**
  for each $a \in \Sigma$ **do**
    if $\mathcal{A}$ has a reset word of length l with prefix wa **then**
      $w := wa$; **break for**
    end if
  end for
end while
return w

computing a shortest reset word: an instance of the former problem is also an instance of the latter problem, and a solution of the latter problem can be turned into a solution of the former problem by computing its length. Q.E.D.

5 Conclusion

We have investigated several decision problems and search problems about finding reset words in finite automata. The results we have obtained shed more light on the difficulty of computing such words. In particular, deciding whether for a given automaton a shortest reset word has length k is DP-complete, and computing the length of a shortest reset word is $\mathrm{FP}^{\mathrm{NP}[\log]}$-complete, i.e. as hard as calculating the maximum number of simultaneously satisfiable clauses of a Boolean formula. A summary of all our results is depicted in Figure 3. (See [7, 10] for the relationships between the referred complexity classes.)

ACKNOWLEDGEMENTS. We thank an anonymous reviewer for pointing out [5]. Moreover, we are grateful to Christof Löding and Wolfgang Thomas for helpful comments on an early draft of this paper.

References

1. M. V. Berlinkov. Approximating the length of synchronizing words. In *Proceedings of the 5th International Computer Science Symposium in Russia, CSR 2010*, volume 6072 of *Lecture Notes in Computer Science*, pages 37–47. Springer-Verlag, 2010.
2. J. Černý. Poznámka k. homogénnym experimentom s konecnými automatmi. *Matematicko-fyzikalny Časopis Slovensk. Akad. Vied*, 14(3):208–216, 1964.
3. J. Černý, A. Pirická, and B. Rosenauerová. On directable automata. *Kybernetica*, 7(4):289–298, 1971.

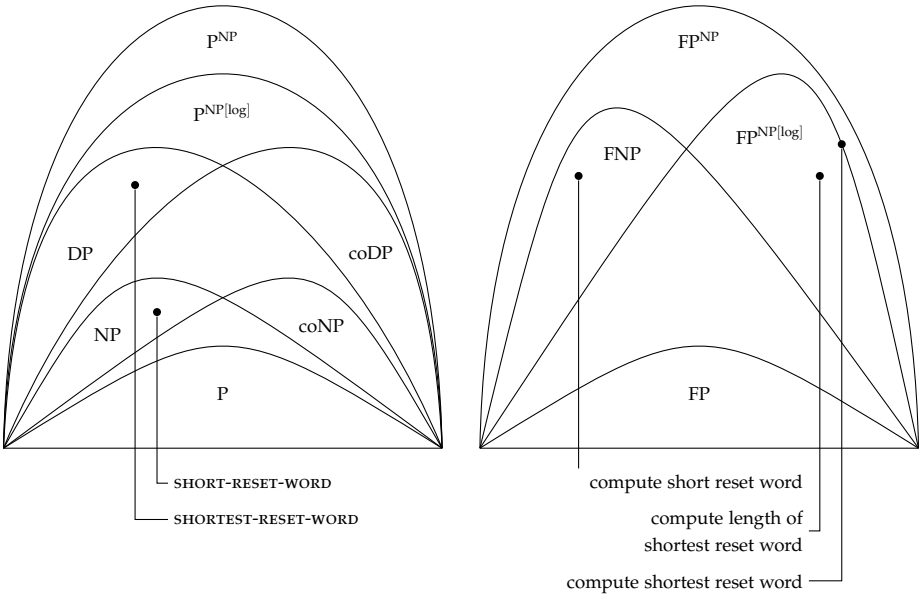

Figure 3. Summary of results.

4. D. Eppstein. Reset sequences for monotonic automata. *SIAM Journal on Computing*, 19(3):500–510, 1990.
5. P. Gawrychowski. Complexity of shortest synchronizing word. Unpublished manuscript, April 2008.
6. M. W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36:490–509, 1988.
7. C. H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.
8. J.-É. Pin. On two combinatorial problems arising from automata theory. *Annals of Discrete Mathematics*, 17:535–548, 1983.
9. W. Samotij. A note on the complexity of the problem of finding shortest synchronizing words. In *Proceedings of AutoMathA 2007*. University of Palermo (CD), 2007.
10. A. L. Selman. A taxonomy of complexity classes of functions. *Journal of Computer and System Sciences*, 48(2):357–381, 1994.
11. L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.
12. M. V. Volkov. Synchronizing automata and the Černý conjecture. In *Proceedings of the 2nd International Conference on Language and Automata Theory and Applications, LATA 2008*, volume 5196 of *Lecture Notes in Computer Science*, pages 11–27. Springer-Verlag, 2008.

A Complexity classes

In this appendix, we want to give an overview on the complexity classes that play a role in this paper. More detailed information can be found in the literature [7, 10].

A.1 Decision problems

We assume that the reader is familiar with the complexity classes P, NP and coNP. The class DP is the closure of NP ∪ coNP under intersection. Equivalently, a language L is in DP if and only if it is of the form $L = L_1 \setminus L_2$ with $L_1, L_2 \in \mathrm{NP}$. The canonical complete problem for DP is the following problem, which is derived from SAT.

> SAT-UNSAT: Given two Boolean formulae φ and ψ in CNF, decide whether φ is satisfiable and ψ is unsatisfiable.

Note that, unless NP = coNP, SAT-UNSAT is not an element of either NP or coNP. Hence, it is conjectured that DP is a proper superclass of NP ∪ coNP. On the other hand, any problem in DP can be solved by a polynomial-time algorithm that has access to an oracle for an NP-complete problem (for instance, SAT). Hence, DP is contained in P^{NP}, the first level of the polynomial hierarchy.

A.2 Counting problems

Formally, a *counting problem* is just a function $F\colon \Sigma^* \to \mathbb{N}$. The class #P consists of all counting problems for which there exists a nondeterministic polynomial-time Turing machine M such that for each input x the number of accepting runs of M on x equals $F(x)$. Analogously to the verifier definition of NP, #P can also be characterised in terms of a relation: we have $F \in \#\mathrm{P}$ if and only if there exists a polynomial-time decidable, polynomially balanced relation $R \subseteq \Sigma^* \times \Sigma^*$ such that $F(x) = |\{y \in \Sigma^* \mid (x, y) \in R\}|$. (A binary relation R is polynomially balanced if there exists a polynomial p such that $|y| \leq p(|x|)$ for all $(x, y) \in R$.)

The simplest (but also most restrictive) kind of a reduction between counting problems is the parsimonious reduction. Formally, a *parsimonious reduction* from a counting problem $F\colon \Sigma^* \to \mathbb{N}$ to another counting problem $G\colon \Sigma^* \to \mathbb{N}$ is a polynomial-time computable function $f\colon \Sigma^* \to \Sigma^*$ such that $|F(x)| = |G(f(x))|$ for all $x \in \Sigma^*$. A function problem F is #P-complete if $F \in \#\mathrm{P}$ and for every $G \in \#\mathrm{P}$ there exists a parsimonious reduction from G to F. (In the literature, #P-hardness is often defined via polynomial-time Turing reductions, which are more general than parsimonious reductions.)

The canonical #P-complete problem is #SAT where the number of satisfying assignments for a given Boolean formula is sought. In fact, #SAT is #P-hard even

for formulae in conjunctive normal form with only two literals per clause. (Note that 2SAT, the restriction of SAT to such formulae, is in P)

A.3 Search problems

A more general concept for computational problems is the one of a search problem. Formally, a *search problem* is a binary relation $R \subseteq \Sigma^* \times \Sigma^*$. If $(x, y) \in R$, we say that y is a *solution* for x (wrt. R). An algorithm A *solves* the search problem R, if the following two conditions hold for every input $x \in \Sigma^*$:

- If x has a solution, then A accepts x and outputs such a solution, i.e. a word $y \in \Sigma^*$ with $(x, y) \in R$.
- If x has no solution, then the algorithm rejects x.

Any decision class that is defined with respect to a deterministic machine model can be extended to a class of search problems in a straightforward way. For example, the class FP consists of all search problems solvable in polynomial time by a deterministic Turing machine with a dedicated output tape. By equipping the machine with an oracle for an NP-complete problem, we obtain the complexity classes $\mathrm{FP}^{\mathrm{NP}}$ and $\mathrm{FP}^{\mathrm{NP[log]}}$, depending on the allowed number of queries. For $\mathrm{FP}^{\mathrm{NP}}$, the number of queries is not restricted, whereas for $\mathrm{FP}^{\mathrm{NP[log]}}$ only a logarithmic number of queries (in the length of the input) is allowed. Obviously, we have $\mathrm{FP} \subseteq \mathrm{FP}^{\mathrm{NP[log]}} \subseteq \mathrm{FP}^{\mathrm{NP}}$.

For the class NP, it turns out that there are two different classes of search problems that can be derived from it. If one takes the usual definition of NP that refers to nondeterministic Turing machines, one arrives at the class NPMV of search problems that can be solved by a nondeterministic machine in polynomial time (see [10] for a formal definition). On the other hand, if one takes the "verifier definition" of NP, one arrives at the class FNP (called NPMV_g in [10]) of search problems where the underlying binary relation R is both polynomially balanced and decidable in polynomial time.

Although both classes FNP and NPMV are derived from the same class NP, they are not necessarily equal. It is easy to see that $\mathrm{FNP} \subseteq \mathrm{NPMV}$, but we have $\mathrm{NPMV} \subseteq \mathrm{FNP}$ if and only if $\mathrm{FP} = \mathrm{FNP}$, which in turn is equivalent to $\mathrm{P} = \mathrm{NP}$ [10]. Regarding other inclusions, both FNP and NPMV are contained in $\mathrm{FP}^{\mathrm{NP}}$, but they are incomparable with $\mathrm{FP}^{\mathrm{NP[log]}}$ under the assumption that $\mathrm{NP} \neq \mathrm{coNP}$ [10].

As for decision and counting problems, one can compare search problems using reductions. However, a reduction between two search problems does not only consist of one function that maps instances to instances but also of another function that maps solutions to solutions (or even a collection of such functions, one for each instance). Formally, a reduction from a search problem $R \subseteq \Sigma^* \times \Sigma^*$ to another search problem $S \subseteq \Sigma^* \times \Sigma^*$ consists of two polynomial-

time computable functions $f\colon \Sigma^* \to \Sigma^*$ and $g\colon \Sigma^* \times \Sigma^* \to \Sigma^*$ such that the following two conditions hold for every $x, y \in \Sigma^*$:

– If x has a solution wrt. R, then $f(x)$ has a solution wrt. S.
– If y is a solution for $f(x)$ wrt. S, then $g(x, y)$ is a solution for x wrt. R.

The canonical complete problem for FNP is FSAT, the problem of computing a satisfying assignment for a given Boolean formula. In fact, FSAT is not only complete for FNP, but also for NPMV. This does not imply P = NP since FNP is *not* closed under reductions unless P = NP: Consider, for instance, the variant FSAT_1 of FSAT where one has to output 1 if the formula is satisfiable instead of outputting a satisfying assignment, i.e. $\text{FSAT}_1 := \{(\varphi, 1) \mid \varphi \in \text{SAT}\}$. Clearly, there exists a reduction from FSAT_1 to FSAT, but FSAT_1 is not contained in FNP unless $\text{SAT} \in \text{P}$.

A problem that is complete for $\text{FP}^{\text{NP}[\log]}$ is MAX-SAT-SIZE, where the task is to compute the maximal number of clauses that are satisfiable simultaneously for a given Boolean formula in conjunctive normal form. Finally, if clauses are equipped with weights and an assignment that maximises the total weight of satisfied clauses is sought, one arrives at MAX-WEIGHT-SAT, a problem that is complete for FP^{NP} [6].